# Programmable Deterministic Built-in Self-test

Abdul-Wahid Hakmi, Hans-Joachim Wunderlich, Christian G. Zoellin
Institut für Technische Informatik
Universitaet Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany

Andreas Glowatz, Friedrich Hapke, Juergen Schloeffel, Laurent Souef
NXP Semiconductors
Georg-Heyken-Str. 1, D-21147 Hamburg, Germany

*Abstract*— **In this paper, we propose a new programmable deterministic Built-In Self-Test (BIST) method that requires significantly lower storage for deterministic patterns than existing programmable methods and provides high flexibility for test engineering in both internal and external test.**

**Theoretical analysis suggests that significantly more care bits can be encoded in the seed of a Linear Feedback Shift Register (LFSR), if a limited number of conflicting equations is ignored in the employed linear equation system. The ignored care bits are separately embedded into the LFSR pattern. In contrast to known deterministic BIST schemes based on test set embedding, the embedding logic function is not hardwired. Instead, this information is stored in memory using a special compression and decompression method. Experiments for benchmark circuits and industrial designs demonstrate that the approach has considerably higher overall coding efficiency than the existing methods.**

*Index Terms*— **Deterministic BIST, Test data compression**

## I. Introduction

Mixed mode BIST schemes use pseudo-random patterns to detect most faults while random pattern resistant faults are specifically targeted by deterministic patterns [1], [2], [3], [4]. The pseudo-random patterns may be generated by running linear feedback shift registers (LFSRs) in autonomous mode. For generating deterministic patterns two approaches are most widespread. Either deterministic patterns are encoded as the seeds of an LFSR [5] or the deterministic patterns are embedded into a pseudo-random sequence using bit-flipping [6], [7] or bit-fixing [8] logic.

The main advantage of reseeding is its programmability as the seeds are stored in a RAM. Many variants have been proposed [9], [10], [11], [12], [13], [14], [15], [16], which try to reduce the LFSR length and storage requirements but the amount of required storage remains significant.

For test set embedding, a combinational logic changes a pseudo-random sequence such that deterministic patterns are embedded. For this, less test information is required compared to reseeding [6] and the combinational logic can be synthesized as a multilevel circuit [7]. But in contrast to LFSR reseeding, a change in the test set requires significant hardware changes because a different combinational logic has to be synthesized. In addition, the embedding functions are often rather irregular and may require significant hardware overhead.

The goal of the method presented here is to combine the advantages of programmable reseeding and hardware efficient test set embedding. An LFSR is reseeded, but unlike conventional reseeding a small number of inconsistent specified bits are ignored during seed computation, which in turn increases encoding efficiency. The ignored bits are then embedded into the LFSR pattern and the corresponding information is stored in compressed form so that programmability is retained. Besides memory and the standard BIST hardware, the only additional on-chip circuitry is a simple decoder, which is not test set dependent. Seeds and embedding information may also be stored in the automatic test equipment or a dedicated chip [17] to implement external BIST.

The organization of the paper is as follows: In section 2 the coding efficiency of the presented method is analytically estimated and compared to existing methods. Section 3 gives an overview of the targeted BIST scheme, and the seed computation procedure is explained in section 4. Section 5 describes the compression and decompression hardware for the embedding information. Finally, experimental results are presented in section 6.

## II. Nearly complete reseeding with full fault coverage

The seed for a deterministic pattern is computed by solving a system of linear equations, where each variable represents

a bit of the LFSR seed and each equation corresponds to one specified bit in the pattern. Some equations may cause the linear equation system to become inconsistent, which prevents certain patterns from being encoded. If the equation system has a solution, the solution determines the seed.

Figure 1 shows a small example of the STUMPS scheme [1], where an 8-bit LFSR feeds 8 scan chains. Here a test vector consists of 8 bits shifted into the scan chains within a single clock cycle, and a test pattern is formed by three vectors.

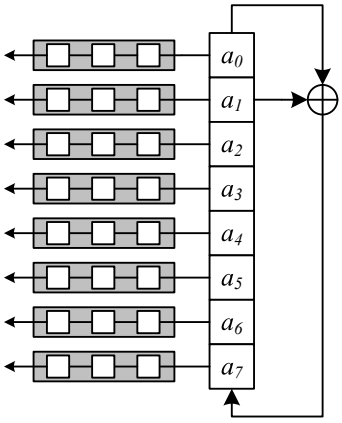


Fig. 1. STUMPS scheme

| $P_1$ | | |
|---|---|---|
| 0 | X | X |
| X | X | X |
| 1 | X | 0 |
| X | X | X |
| X | X | X |
| 0 | X | X |
| X | 1 | X |
| X | 0 | 1 |

$\Rightarrow$

$$\begin{aligned} a_0 &= 0 \\ a_2 &= 1 \\ a_4 &= 0 \\ a_5 &= 0 \\ a_7 &= 1 \\ a_0 + a_1 &= 0 \\ a_1 + a_2 &= 1 \end{aligned}$$

$\Rightarrow$

| $S_1$ |
|---|
| 0 |
| 0 |
| 1 |
| X |
| 0 |
| 0 |
| X |
| 1 |

(a) Solvable system

| $P_2$ | | |
|---|---|---|
| X | X | 0 |
| 0 | X | 1 |
| X | 1 | X |
| 0 | X | 1 |
| 0 | X | X |
| X | 1 | 0 |
| 1 | X | X |
| X | 1 | 1 |

$\Rightarrow$

$$\begin{aligned} a_3 &= 0 \\ a_3 &= 1 \\ a_4 &= 0 \\ a_5 &= 1 \\ a_6 &= 1 \\ a_7 &= 0 \\ a_0 + a_1 &= 1 \\ a_1 &= 0 \\ a_2 &= 0 \\ a_1 + a_2 &= 1 \end{aligned}$$

$\Rightarrow$

| $S_2$ |
|---|
| No Seed |

(b) Non-solvable system

Fig. 2. Example of seed computation

In order to compute a seed for a deterministic pattern $P_1$, we represent each bit of $P_1$ by an equation in terms of the seed variables $a_0, a_1, \ldots, a_7$ as shown in figure 2(a). The system of equations may become linearly dependent and not solvable any more, pattern $P_2$ in figure 2(b) is an example of this. If a phase shifter is inserted, the system of equations may become more complex, but the principle will not change. In [5], it was shown that the probability of an unsolvable equation system for an arbitrary pattern can be reduced to $10^{-6}$ if the length of the LFSR is $s_{max} + 20$ where $s_{max}$ is the maximum number of specified bits. If the equations $a_3 = 0$ and $a_1 + a_2 = 1$ in the equation system of $P_2$ are ignored, the system becomes consistent and a seed can be computed.

In the sequel, we use the probabilistic model developed in [8], [9] in order to estimate the impact of ignoring a certain number of equations. There, the probability of having a consistent linear equation system for $s$ specified bits and an LFSR of length $k$ is given as $P_{seed}(k, s)$. These values are determined recursively, and also listed in a table. The probability of encountering an inconsistent equation system is:

$$P_{noseed}(k, s) = 1 - P_{seed}(k, s)$$

For a given $b$, any combination of $b$ out of $s$ equations is a candidate to be ignored. Let $P_{noseed}(k, s, b)$ be the probability, that we cannot find $b$ equations such that we are able to encode the remaining $s - b$ bits. The number of all combinations of $b$ bits is $\frac{s!}{b!(s-b)!}$, and the probability that all the resulting systems of equations are not solvable is

$$P_{noseed}(k, s, b) = (P_{noseed}(k, s - b))^{\frac{s!}{b!(s-b)!}}$$

Figure 3 shows the set of curves $P_{noseed}(k, s, b)$ as the function of $s$ with parameters $k = 128$ and $b = \{0, 1, 2, 3\}$. Here, $P_{noseed}(128, s, 0)$ represents the probability for the conventional reseeding in [5]. Obviously, by ignoring a small number

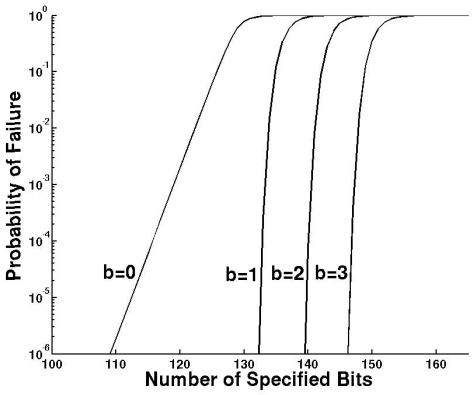


Fig. 3. Values of $P_{noseed}(k, s, b)$ for $k = 128$

of equations, a seed can be computed for a significantly larger number of specified bits compared to conventional reseeding. The largest gain is achieved by ignoring a single equation and the coding efficiency increases almost linearly if further equations are ignored.

In the example of Figure 3, we are able to save approximately 20 bits of a seed compared with conventional reseeding if we are allowed to ignore one equation. However, the ignored bits have to be encoded by their position in the scan chain. Hence, a rough estimation of the savings in this case is $20 - log(n)$, where $n$ is the pattern length.

In practice, additional savings are obtained for two reasons:

- For the ignored equations, the corresponding value in the pattern generated by the LFSR will assume the opposite value. In some cases, this pattern still detects the target fault, an effect that is exploited in test set compaction [18], and the LFSR pattern may remain unchanged.
- For parallel scan chains, the embedding information can even be shared between the scan chains, thereby further increasing coding efficiency.

The next section presents the BIST scheme implementing nearly complete reseeding with corrections. The technique may also be combined with multiple polynomial LFSR reseeding [9].

## III. The Target BIST Scheme

During seed computation of nearly complete reseeding, one or more bits of a specified vector may raise conflicts and will be ignored. Addressing the ignored bits of a specified vector individually may introduce significant overhead in hardware and test time. Hence, we address a complete vector if any of its bits causes a conflict during seed computation.

When decoding a computed seed, the LFSR will generate the desired values at compatible bit positions and opposite values at conflicting bit positions. In order to retain the values at compatible bit positions and to flip the opposite values, a *flip vector* is assigned to each ignored vector. The seeds, the ignored positions and the flip vectors are stored in memory. In order to reduce the memory requirements, ignored positions are encoded as the distance between two ignored positions, and flip vectors are compressed using a special method described below. A small decoder is then used to generate the flip vectors. Figure 4 shows an overview of the target BIST scheme.

Two main approaches to seed computation are popular in current literature. Either a complete pattern is encoded in a single seed [5], [9], [10], [11], [12], or continuous subsequences of a single or multiple patterns are encoded in each seed [16]. For the latter approach, it was shown that storage requirements are improved and shorter LFSRs can be employed [16]. Although nearly complete reseeding can be employed with any of the two reseeding techniques, the second approach has been selected here, as multiple patterns can be encoded before the number of ignored vectors exceeds a predefined limit. Furthermore, control information for the start of the next seed can be derived implicitly by starting the next seed just after the last ignored position associated with a seed. This control

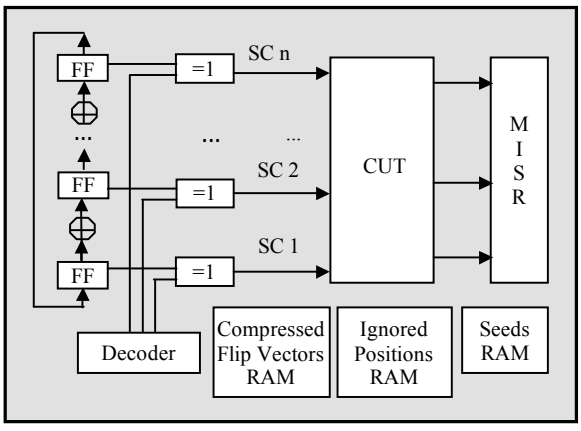


Fig. 4. Overview of the target BIST scheme

information had to be stored separately in [16].

## IV. Seed Computation

Since a mixed-mode BIST is implemented, fault simulation is performed for some pseudo-random patterns to identify random testable faults. Deterministic patterns are generated for the remaining faults and the resulting test set is compacted by removing patterns which are contained in some other pattern. For seed computation, a limit $b$ is set that defines the number of vectors that are allowed to be ignored for each seed. Since continuous reseeding is used, blocks of a certain number of patterns have to be created and encoded by seeds. The seeds are decoded again into test patterns which do not contain unspecified values 'X' anymore. Fault simulation is performed with these completely specified patterns, and not yet encoded patterns are dropped if all of their target faults have been detected.

Figure 5 shows the flow chart of seed computation. Seed computation is done in an iterative manner. It starts with empty sets of encoded and ignored vectors. A specified vector is added to the set of encoded vectors, the linear equation system is generated and it is established whether or not a solution exists. If it does, the process continues with the next specified vector. Otherwise, the current vector is moved from the encoded set to the ignored set first. The process of adding a specified vector into the encoded or ignored set continues until the number of ignored vectors exceeds the defined limit $b$.

When the addition of a certain specified vector makes the linear equation system unsolvable, most times, the current vector is not the only choice to be ignored in order to make the set solvable again. The ignored vector can be chosen from a number of possible candidates and choosing another vector instead of the most recent one, might result in fewer conflicts later on. But as the best choice of ignored positions might change with every new vector, the current vector is chosen as the victim until the allowed limit is crossed. If $n$ is the total number of vectors considered so far, then as soon as the defined limit $b$ is exceeded, the ignored set contains $b+1$ vectors and the encoded set contains $n-b-1$ vectors. At this point, a search is executed for a combination of $b$ vectors such that ignoring these $b$ instead of $b+1$ vectors, causes the other $n-b$ vectors to be encodable. Linear equation solvers for boolean linear equation systems can be efficiently implemented using bit-parallelism and checking all possible equation systems for all subsets of $n-b$ vectors is feasible.

If a subset of vectors is found for which the system of linear equations is consistent, the sets of encoded and ignored vectors are updated and the process continues. If no such subset exists, the current vector is used to delimit the start of the next seed, which means that $b+1$ vectors are actually ignored for each seed instead of $b$ vectors . The $b+1^{th}$ ignored position is different from the first $b$ ignored positions in that no new vector is encoded after it, but a new seed is started. Using this technique, the test control is simplified and encoding efficiency is improved as no additional run length has to be

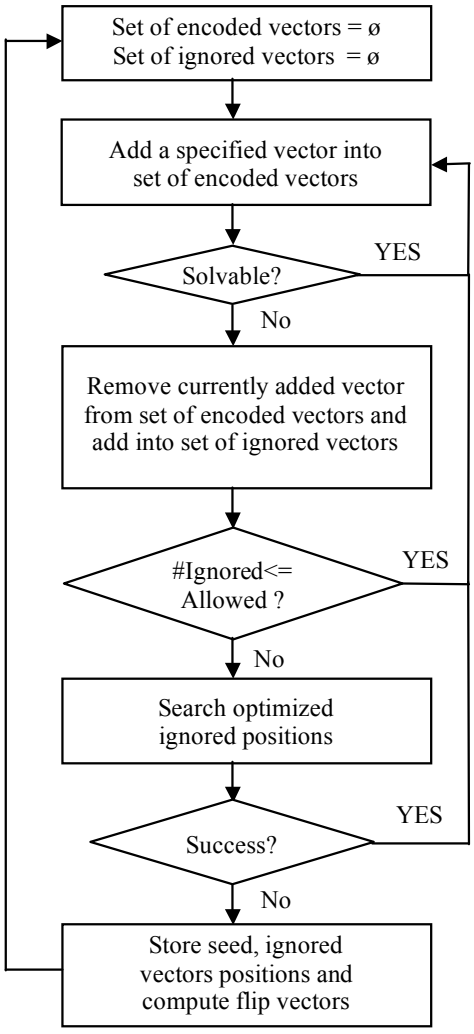


Fig. 5. Seed Computation Flowchart

stored. Before starting the next iteration with empty sets of encoded and ignored vectors, the flip vectors are derived from the ignored vectors and are stored together with the seed and ignored positions. By simulating the unchanged pattern (i.e. without flip vectors applied) we can determine if this information is required for fault detection. This fault simulation is done only w.r.t. the target faults of the pattern and does not add significant computational overhead.

*a) Example:* Suppose the 5 test patterns shown in figure 6 have to be encoded, which include the two patterns in the example of figure 2, and we are allowed to ignore 2 specified vectors per seed. The seed computation procedure starts by adding the 1st vector of $P_1$ (left most vector) to the encoded set. The equations are generated and it is determined that this equation system has a solution. Then, the 2nd vector is added and equations for both vectors in the set are generated to observe solvability. The encoded set remains solvable until the 4th vector is added. As the limit is not yet exceeded this vector is moved to the set of ignored vectors. Then the 6th and 8th vectors are found to cause the equations to be unsolvable, so they are moved to the ignored set, too. By ignoring the 8th vector, the size of the ignored set becomes 3, which is greater than allowed. Now it is attempted to optimize the selection of ignored vectors so that instead of three vectors only two vectors have to be ignored and the rest of the vectors become solvable. For this we generate equations and check solvability of all the possible 6 out of 8 vectors. We find that none of these subsets are solvable, so the seed of the encoded set $1, 2, 3, 5, 7$ is saved. In order to derive flip vectors for the ignored set, the location of the bits related to the conflicting equations are determined and the flip vector is assigned '0' for matching bits, '1' for conflicting bits and 'X' for the rest. The remaining seven vectors are encoded by the second seed using the same method. Figure 7 shows both seeds along with their ignored positions and associated flip vectors.

## V. Compression and Decompression of Embedding Information

The embedding information contains flip positions and flip vectors associated with them. The flip positions are efficiently encoded by encoding the distances between two consecutive flip positions instead of their absolute numbers. These distances are used during test to identify the ignored vectors and start of the next seed.

For the flip vectors, it is exploited that most of the flip vectors contain only few specified bits. A number of test data compression and decompression techniques [19], [20], [21], [22], [23], [24], [25] are available that can be used for flip vectors, but all of these are optimized to compress and decompress complete patterns. For the relatively short flip vectors, test time would be prolonged without achieving any significant storage reduction with these methods. A special compression and decompression method has been developed for flip vectors which is similar to the STARBIST method [26] and offers significant storage reduction.

### A. Compression Algorithm

The main idea of the compression algorithm of flip vectors is that a reference vector is computed with which the

| $P_1$ | | | $P_2$ | | | $P_3$ | | | $P_4$ | | | $P_5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | X | X | X | X | 0 | X | X | 1 | 1 | X | X | 1 | X | X |
| X | X | X | 0 | X | 1 | 1 | 0 | 1 | 1 | 0 | 0 | X | 1 | 1 |
| 1 | X | 0 | X | 1 | X | 0 | 0 | X | X | 0 | X | 0 | X | X |
| X | X | X | 0 | X | 1 | 1 | 1 | X | X | 0 | 1 | X | 0 | 1 |
| X | X | X | 0 | X | X | X | X | 0 | X | 1 | 0 | 1 | X | X |
| 0 | 1 | X | X | 1 | 0 | X | X | 0 | X | X | 0 | 0 | X | X |
| X | 1 | X | 1 | X | X | X | X | X | X | X | X | X | 0 | 0 |
| X | 0 | 1 | X | 1 | 1 | 1 | X | 0 | 1 | X | 1 | 0 | X | X |

Fig. 6. Set of 5 Patterns

| $S_1$ | $F_1$ | | | $S_2$ | $F_2$ | | |
|---|---|---|---|---|---|---|---|
| | 4 | 6 | 8 | | 2 | 5 | 7 |
| 0 | X | 0 | X | 1 | 0 | 1 | X |
| 0 | 0 | 0 | 0 | 1 | 0 | X | 1 |
| 1 | X | X | 1 | 1 | X | 1 | X |
| 1 | 1 | 1 | 1 | 0 | X | X | 1 |
| 0 | 1 | X | X | 0 | X | 1 | X |
| 0 | X | 0 | X | 0 | X | 0 | X |
| 1 | 0 | X | X | 1 | X | X | 0 |
| 1 | X | 1 | X | 0 | 1 | 0 | X |

Fig. 7. Computed seeds and flip vectors

flip vectors have a minimum number of conflicts. In most cases, a large number of the flip vectors are fully compatible, and the ignored position and flip vector is encoded as a tuple $(i, 0)$ where $i$ is the address of the ignored vector relative to the previous ignored. Otherwise, the encoding is $(i, 1, (p_0, 0) \ldots (p_{n-1}, 0), (p_n, 1))$, where $p_0 ... p_n$ are the positions in which the flip-vector differs from the reference vector.

The reference vector contains a '1' at a certain bit position, if the flip vectors have more '1's than '0's, otherwise it contains a '0'. Figure 8 shows the compression of the flip vectors computed in the previous example. Each flip vector is a column of a matrix, and we determine which of '0' or '1' has the highest frequency in each row. This value is entered in the reference vector RV. In the example, we find that the first row has more '0's than '1's, so the first bit of the reference vector RV is assigned a zero. The complete reference vector is computed this way and is shown in the column next to the flip vectors.

Figure 8 also illustrates the encoding of the ignored positions and the flip vectors. A comparison of the flip vectors with the reference vector reveals that the first 4 flip vectors are fully compatible, while the last two vectors have conflicts. So, the first 4 flip vectors are encoded by storing a single status bit '0' with the ignored positions. For the second last vector, bit number 0 and bit number 7 have conflicts with the reference vector. This vector is encoded as $(3, 1, (0, 0), (7, 1))$ which indicates that the reference vector can be used as the flip vector after modifying the bit 0 and 7. The status bit '1' in $(7, 1)$ means that the reference vector is ready to be used after this modification.

Using the proposed technique the test set of figure 6, containing 5 patterns and 53 specified bits, can be encoded with a total of 54 bits where 16 bits are required to store seeds, 12 bits to store ignored positions and 26 bits to store flip vectors. The same patterns would have required 210 bits and 90 bits in case of conventional [5] and variable pattern length [16] reseeding techniques respectively.

### *B. Decompression Architecture*

To decode the encoded flip vectors, we need the reference vector, the information whether the flip vector of an ignored position is compatible with the reference vector, and the conflicting positions. Figure 9 shows the architecture of the flip vector generator. It contains a register to store the reference vector (RV Register), a modify unit to generate incompatible flip vectors and a multiplexer to select between the reference and incompatible flip vector depending on the status bit of the ignored position.

The basic idea for the Modify Unit is taken from the decompression architecture proposed in [17] and is adapted to the new requirements. It creates flip vectors serially during shifting by using the reference vector in the RV Register and the encoded conflict positions in the Compressed Flip Vectors RAM. On receiving the *start* signal, the Incompatible Flip Vec-

| $F_1$ | | | $F_2$ | | | $RV$ |
|---|---|---|---|---|---|---|
| 4 | 6 | 8 | 2 | 5 | 7 | |
| X | 0 | X | 0 | 1 | X | 0 |
| 0 | 0 | 0 | 0 | X | 1 | 0 |
| X | X | 1 | X | 1 | X | 1 |
| 1 | 1 | 1 | X | X | 1 | 1 |
| 1 | X | X | X | 1 | X | 1 |
| X | 0 | X | X | 0 | X | 0 |
| 0 | X | X | X | X | 0 | 0 |
| X | 1 | X | 1 | 0 | X | 1 |

$C_1 = \{(4, 0), (2, 0), (2, 0)\}$

$C_2 = \{(2, 0), (3, 1, (0, 0), (7, 1)), (2, 1, (1, 1))\}$

Fig. 8. Example of flip vector compression

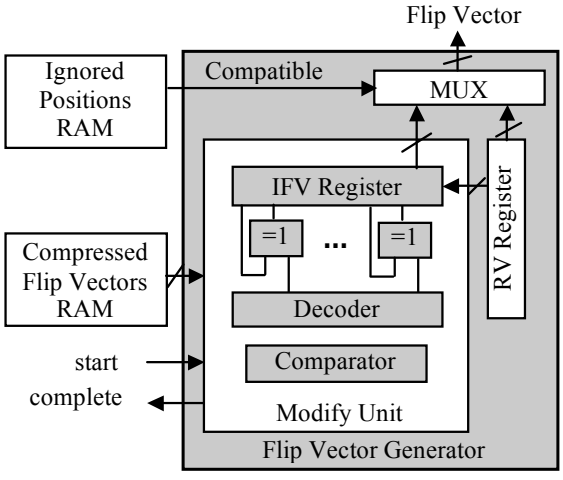


Fig. 9. Decompression Hardware

tor Register (IFV Register) is reset to the reference vector and the conflicting positions are read from the RAM and flipped in the IFV Register until the comparator detects that the status bit of a conflicting position is '1'. The *complete* signal is set to high at this point, indicating that the flip vector is ready to be used. The number of clocks needed to produce a complete incompatible vector is equal to the number of conflicts between an incompatible flip vector and the reference vector.

Here it should be noted that the Modify Unit is completely independent and it does not need to wait until the ignored position of an incompatible flip vector is met. It produces the next incompatible flip vector in advance and waits for its use. As ignored positions are randomly distributed and very few flip vectors require modification the decompression does not cause any noticeable speed degradation in general. Some cycles could be wasted only in the case if there are two consecutive flip positions and both of them have incompatible flip vectors, an extremely rare case. Moreover, the flip vector generator only depends on the number of scan chains in the circuit and is independent of the test set or size of the circuit. For example, the size of the flip vector generator is 1507 NAND gates for any circuit with 64 scan chains.

## VI. Experimental results

In order to validate the efficiency of the presented algorithm and test architecture, experiments were performed on circuit models from the ISCAS89 and ITC99 collections of benchmark circuits, as well as considerably larger industrial circuits from NXP. The method itself was implemented in C++. The results are compared to the variable pattern length per seed technique known as continuous reseeding [16], which is the most efficient single polynomial technique published so far and is equivalent to evaluating the presented algorithm using $b = 0$.

Table I shows the important properties of the employed circuits and lists the hardware overhead of the proposed scheme. The first column contains the names of the circuits, which have a leading s and b for the ISCAS and ITC benchmark circuits and start with a leading p for the industrial circuits. The next column reports the total number of primary inputs and flip-flops in the circuit. The third column contains the number of scan chains in each circuit. An arbitrary scan chain configuration was used for benchmark circuits while the experiments for the industrial circuits were conducted using the provided configurations. The column labeled 'LFSR' contains the length of the LFSR. A phase shifter with two to three taps for each scan chain was used in conjunction with each LFSR. Finally, the number of two-input gates in each circuit is followed by the hardware overhead for the proposed scheme in gates. This includes the overhead caused by the flip vector generator, the flipping logic and the memory access mechanism to write and read data from RAM, assuming that all the information is stored on chip using bit addressable memories.

From Table I, it is obvious that the hardware overhead associated with the proposed scheme is quite small compared to the size of the circuit under Test. Especially for large circuits this overhead can be considered insignificant.

| Circuit Name | # PIs + PPIs | # Scan Chains | LFSR | Circuit # gates | HW Overhead # gates | # r.p.r. Faults | # Specified bits |
|---|---|---|---|---|---|---|---|
| s9234 | 247 | 4 | 32 | 6,045 | 507 | 913 | 5,762 |
| s13207 | 700 | 10 | 32 | 8,741 | 647 | 1,109 | 3,465 |
| s15850 | 611 | 9 | 32 | 10,308 | 648 | 780 | 5,932 |
| s38417 | 1,664 | 20 | 32 | 24,079 | 948 | 3,141 | 22,816 |
| s38584 | 1,464 | 18 | 32 | 22,092 | 838 | 706 | 3,592 |
| b17 | 1,452 | 18 | 32 | 37,446 | 954 | 23,558 | 133,822 |
| b22 | 767 | 12 | 32 | 33,569 | 842 | 6,639 | 106,206 |
| p286k | 18,351 | 55 | 128 | 332,726 | 1,960 | 56,252 | 436,003 |
| p330k | 18,010 | 64 | 128 | 312,666 | 2,163 | 68,736 | 518,161 |
| p388k | 25,005 | 50 | 128 | 433,331 | 1,860 | 31,455 | 222,707 |
| p418k | 30,430 | 64 | 128 | 382,633 | 2,083 | 71,630 | 359,260 |
| p951k | 91,994 | 82 | 64 | 816,072 | 2,472 | 55,987 | 407,361 |

Table I. Circuit statistics and properties

Table I also shows the number of random pattern resistant faults, which were determined by fault simulation of $10,000$ random patterns. A compacted test set with partially specified patterns was generated for these faults. The total number of specified bits in the compacted unencoded test set is given in the last column of Table I.

First, an experiment was conducted evaluating the coding efficiency as a function of the number of ignored vectors per seed. The results of this experiment are reported in Table II and the total storage requirements are given for $b = 1$, 2 and 3. Obviously, for the majority of the circuits the highest gain is achieved by ignoring two vectors per seed. Consequently, $b = 2$ was used as the limit in all other experiments.

Table III compares the storage requirements between the presented nearly complete reseeding and the technique in [16]. The fault coverage was not sacrificed for both methods. After the column containing the name of the circuits, the results for each method are provided in two blocks, first for the complete reseeding ($b = 0$) and then for the incomplete reseeding with complete fault coverage and $b = 2$. Each block is divided into two columns. The first of these columns lists the storage requirements in terms of bits. For both methods, this includes the amount of memory required to store seeds and all the required control information. The second column for each method reports the encoding efficiency computed as:

$$\text{Enc. Eff.} = \frac{\text{\# specified bits in unencoded test set}}{\text{\# total bits in encoded test set}}$$

In the last column of Table III, the encoding efficiency of the two methods is compared and the relative improvement for the proposed method is given in percent.

For p951k, the presented algorithm achieved marginally better results when using an LFSR of width 64. It is obvious that an LFSR of 64 bit is not sufficient to encode all occurring vectors, because each vector is 82 bits wide. In conclusion, not all patterns can be encoded using continuous reseeding. For the nearly complete reseeding, these vectors are encoded as flip-vectors and complete fault coverage can be retained. For comparison purposes: When combined with an LFSR of width 128, continuous reseeding achieves an encoding efficiency of 0.64.

| Circuit Name | $b = 1$ Bits | $b = 2$ Bits | $b = 3$ Bits |
|---|---|---|---|
| s9234 | 7,918 | **7,468** | 7,572 |
| s13207 | 4,426 | 4,321 | **4,180** |
| s15850 | 7,497 | **6,960** | 7,184 |
| s38417 | 32,284 | 29,498 | **28,932** |
| s38584 | 4,725 | **4,494** | 4,597 |
| b22 | 117,557 | **106,917** | 110,137 |

Table II. Storage requirements using different number of ignored vectors

| Circuit Name | Continuous reseeding Bits | Continuous reseeding Eff. | Proposed approach Bits | Proposed approach Eff. | Improvement % |
|---|---|---|---|---|---|
| s9234 | 9,480 | 0.61 | 7,468 | 0.77 | 27% |
| s13207 | 5,800 | 0.60 | 4,321 | 0.80 | 34% |
| s15850 | 10,360 | 0.47 | 6,960 | 0.85 | 49% |
| s38417 | 51,870 | 0.44 | 29,498 | 0.77 | 76% |
| s38584 | 7,280 | 0.49 | 4,494 | 0.80 | 62% |
| b17 | 243,560 | 0.55 | 112,779 | 1.19 | 116% |
| b22 | 229,710 | 0.46 | 106,917 | 0.99 | 115% |
| p286k | 1,188,728 | 0.37 | 720,135 | 0.61 | 65% |
| p330k | 1,071,018 | 0.48 | 753,777 | 0.69 | 42% |
| p388k | 644,265 | 0.35 | 383,786 | 0.58 | 68% |
| p418k | 1,188,880 | 0.30 | 709,741 | 0.51 | 68% |
| p951k | N/A | N/A | 390,441 | 1.04 | - |

Table III. Comparison of encoding efficiencies

In [27] it was shown analytically that the maximum encoding efficiency that can be achieved by complete reseeding is 1. Even for the best solutions proposed until now, the efficiency observed in reality did not exceed this theoretical limit. It is evident that the encoding efficiency of the proposed method is significantly closer to this theoretical value, even exceeding it for some of the circuits.

## VII. Conclusion

A deterministic BIST method has been presented that is programmable and provides much higher coding efficiency compared to the regular reseeding methods. A special compression and decompression method efficiently encodes and decodes the care bits that could not be encoded in the seed. The use of nearly complete reseeding allows for encoding more care bits into a single seed and improves the coding efficiency without reducing fault coverage. A comparison with the best previously known method shows a significant reduction in test data volume by 20-50% for both benchmark and industrial circuits.

## ACKNOWLEDGMENT

This work was supported by the DFG project Wu245/5-1 and by the BMBF project MAYA. We would like to thank Guenter Bartsch, Stefan Holst, and Michael Imhof from the University of Stuttgart for providing support with the ITI software framework.

## REFERENCES

[1] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*. John Wiley & Sons, 1987.

[2] S. B. Akers and W. Jansz, "Test set embedding in a built-in self-test environment," in *Proceedings International Test Conference 1989, Washington, D.C., USA, August 1989*, 1989, pp. 257–263.

[3] R. Dandapani, J. H. Patel, and J. A. Abraham, "Design of test pattern generators for built-in test," in *Proceedings International Test Conference 1984, Philadelphia, PA, USA, October 1984*, 1984, pp. 315–319.

[4] W. Daehn and J. Mucha, "Hardware test pattern generation for built-in testing," in *Proceedings International Test Conference 1981, Philadelphia, PA, USA, October 1981*, 1981, pp. 110–120.

[5] B. Koenemann, "LFSR-coded test patterns for scan designs," in *Proceedings of the European Test Conference, Munich, Germany*, 1991, pp. 237–242.

[6] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *International Conference on Computer-Aided Design (ICCAD), November 10-14, 1996, San Jose, CA*, 1996, pp. 337–343.

[7] V. Gherman, H.-J. Wunderlich, H. P. E. Vranken, F. Hapke, M. Wittke, and M. Garbers, "Efficient pattern mapping for deterministic logic BIST," in *Proceedings 2004 International Test Conference (ITC 2004), October 26-28, 2004, Charlotte, NC, USA*, 2004, pp. 48–56.

[8] N. A. Touba and E. J. McCluskey, "Altering a pseudo-random bit sequence for scan-based BIST," in *Proceedings IEEE International Test Conference 1996, Washington, DC, USA, October 20-25, 1996*, 1996, pp. 167–175.

[9] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Trans. Computers*, vol. 44, no. 2, pp. 223–233, 1995.

[10] J. Rajski, J. Tyszer, and N. Zacharia, "Test data decompression for multiple scan designs with boundary scan," *IEEE Trans. Computers*, vol. 47, no. 11, pp. 1188–1200, 1998.

[11] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial LFSR reseeding," in *Proceedings IEEE International Test Conference 2001, Baltimore, MD, USA, 30 October - 1 November 2001*, 2001, pp. 885–893.

[12] A. A. Al-Yamani, S. Mitra, and E. J. McCluskey, "BIST reseeding with very few seeds," in *21st IEEE VLSI Test Symposium (VTS 2003), 27 April - 1 May 2003, Napa Valley, CA, USA*, 2003, pp. 69–76.

[13] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test application time and volume compression through seed overlapping," in *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, 2003, pp. 732–737.

[14] C. V. Krishna and N. A. Touba, "3-stage variable length continuous-flow scan vector decompression scheme," in *22nd IEEE VLSI Test Symposium (VTS 2004), 25-29 April 2004, Napa Valley, CA, USA*, 2004, pp. 79–86.

[15] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding," in *Proceedings 10th Asian Test Symposium*, 2001, pp. 325–330.

[16] E. H. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *21st IEEE VLSI Test Symposium (VTS 2003), 27 April - 1 May 2003, Napa Valley, CA, USA*, 2003, pp. 232–240.

[17] A. W. Hakmi, H.-J. Wunderlich, V. Gherman, M. Garbers, and J. Schlöffel, "Implementing a scheme for external deterministic self-test," in *23rd IEEE VLSI Test Symposium (VTS 2005), 1-5 May 2005, Palm Springs, CA, USA*, 2005, pp. 101–106.

[18] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A method to generate compact test sets for combinatorial circuits." in *Proceedings IEEE International Test Conference 1991, Nashville, TN, USA, October 26-30, 1991*, 1991, pp. 194–203.

[19] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan vector compression/decompression using statistical coding," in *17th IEEE VLSI Test Symposium (VTS '99), 25-30 April 1999, San Diego, CA, USA*, 1999, pp. 114–120.

[20] K. Chakrabarty, B. T. Murray, and V. Iyengar, "Built-in test pattern generation for high-performance circuits using twisted-ring counters," in *17th IEEE VLSI Test Symposium (VTS '99), 25-30 April 1999, San Diego, CA, USA*, 1999, pp. 22–27.

[21] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," in *2002 Design, Automation and Test in Europe Conference and Exposition (DATE 2002), 4-8 March 2002, Paris, France*, 2002, pp. 604–611.

[22] F. G. Wolff and C. A. Papachristou, "Multiscan-based test compression and hardware decompression using LZ77," in *Proceedings IEEE International Test Conference 2002, Baltimore, MD, USA, October 7-10, 2002*, 2002, pp. 331–339.

[23] S. Reda and A. Orailoglu, "Reducing test application time through test data mutation encoding," in *2002 Design, Automation and Test in Europe Conference and Exposition (DATE 2002), 4-8 March 2002, Paris, France*, 2002, pp. 387–395.

[24] A. Khoche, E. H. Volkerink, J. Rivoir, and S. Mitra, "Test vector compression using EDA-ATE synergies," in *20th IEEE VLSI Test Symposium (VTS 2002), Without Testing It's a Gamble, 28 April - 2 May 2002, Monterey, CA, USA*, 2002, pp. 97–102.

[25] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain designs," in *20th IEEE VLSI Test Symposium (VTS 2002), Without Testing It's a Gamble, 28 April - 2 May 2002, Monterey, CA, USA*, 2002, pp. 103–110.

[26] K.-H. Tsai, S. Hellebrand, J. Rajski, and M. Marek-Sadowska, "STAR-BIST: Scan autocorrelated random pattern generation." in *Proceedings of the 34th Design Automation Conference (DAC), Anaheim, CA, USA, June 9-13, 1997*, 1997, pp. 472–477.

[27] K. J. Balakrishan and N. A. Touba, "Relating entropy theory to test data compression," in *Proceedings IEEE European Test Symposium*, 2004, pp. 187–192.